(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 July 2001 (26.07.2001)

PCT

(10) International Publication Number
WO 01/53948 A2

(51) International Patent Classification[7]: G06F 11/34

(21) International Application Number: PCT/US01/01745

(22) International Filing Date: 17 January 2001 (17.01.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/176,710	18 January 2000 (18.01.2000)	US
09/651,351	29 August 2000 (29.08.2000)	US

(71) Applicant: ISOGON CORPORATION [US/US]; 330 Seventh Avenue, New York, NY 10001 (US).

(72) Inventors: VARDI, David; Isogon Corporation, 330 Seventh Avenue, New York, NY 10001 (US). HELLBERG, Per; Isogon Corporation, 330 Seventh Avenue, New York, NY 10001 (US). YOUCHNOW, Steven; Isogon Corporation, 330 Seventh Avenue, New York, NY 10001 (US). BARRITZ, Robert; Isogon Corporation, 330 Seventh Avenue, New York, NY 10001 (US).

(74) Agents: MOSKOWITZ, Max et al.; Ostrolenk, Faber, Gerb & Soffen, LLP, 1180 Avenue of the Americas, New York, NY 10036 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

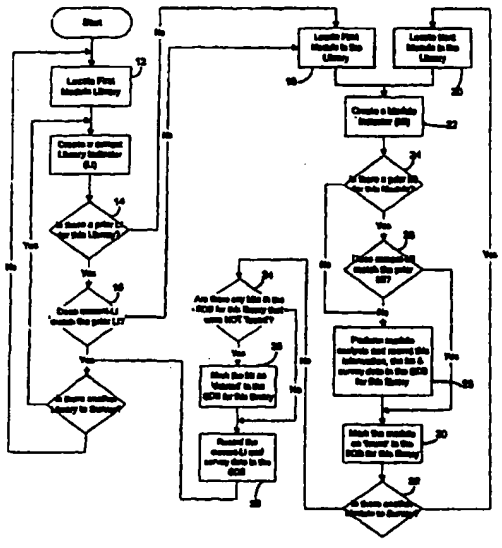
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: CONTINUOUS SURVEYOR MONITOR



(57) Abstract: A method and apparatus for surveying the storage devices of a computer system to generate a list of module names stored on the storage devices in order to produce an inventory of the module names. The inventorying/ surveying process is carried out on a continuous basis so that virtually at all times the list of module names is up to date with the current status of the modules that are stored on the storage devices. The surveying is carried out by one of a variety of processes including constant surveying of module names stored by comparison to an immediately preceding survey. Another approach carries out the surveying by monitoring usage of load modules. A third approach interacts by creating hooks and vectors with the software that is responsible for recording the addition, deletion, or modification of modules on the computer system.

WO 01/53948 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

CONTINUOUS SURVEYOR MONITOR

BACKGROUND OF THE INVENTION

The present invention relates to a method and apparatus for inventorying and surveying the computer software products installed on a computer and, more particularly, the invention relates to a continuous inventorying, surveying and monitoring system and method for inventorying, surveying, and, optionally, monitoring the frequency of use of a variety of computer programs and software products.

Licensed software products, such as those from IBM, Computer Associates or Microsoft, are typically composed of a number of discrete executable components: *exe*-files, batch files, etc., herein collectively referred to as modules. A typical mainframe computer might have 500 products, composed of 500,000 modules on 3,000 libraries, often with many of the products duplicated on a number of libraries. Even a personal computer may contain several thousand exe-files, located in dozens or hundreds of folders.

Isogon is the assignee of the present invention. Prior Isogon patents have described techniques for performing software auditing, including the steps of Surveying (scanning hard-drives or disk storage for modules), Identification (deciding, for each module on each library, what software product it belongs to) and

Monitoring (intercepting and recording module executions). As described in those patents, and as practiced by Isogon's software auditing product, SoftAudit, the steps of Surveying, Identification, and Monitoring are both interrelated and separate processes.

The contents of one prior Isogon patent, namely, the contents of U.S. Patent No. 5,590,056, are incorporated by reference herein.

The Surveying (SURVEYOR) and Identification (IDENTIFIER) functions are batch processes that are performed periodically, while the Monitoring activity (MONITOR) is typically continuous. The SURVEYOR builds a Survey DataBase (SDB) containing a record of every module that it finds on any of the computer's hard drives, indicating the library on which each one was found (recognizing that libraries with the same name but stored on different disk devices are in fact different libraries), and optionally including other identifying information (i.e., a "fingerprint") about the module as well. Such "fingerprint" information typically includes the size of the module, the name and size of constituent sets or parts of the module, the dates that the constituent parts were compiled, the date that the constituent parts were linkedited or bound together to create the module, certain textual data contained within the module, etc. (We refer to the process of collecting this aforesaid information

for a particular module as performing a "module analysis".) Identification is subsequently performed by correlating the data in the SDB with data in a Knowledge Base (KB) that relates module names to product names, by applying certain matching rules and heuristics. Data produced by the MONITOR is not used by either the SURVEYOR or IDENTIFIER.

This procedure has certain deficiencies. To perform a survey, the directory of each library on the computer system must be read, which yields the names of each module in the library. Then, in order to gather the fingerprint information, it is often necessary to read the modules themselves.

Accordingly, surveying an entire computer system is very high in overhead. Performing module analysis requires reading all the data of the module itself, which may comprise several hundred thousand characters of data for each of the half-million or more modules on a system. Surveying takes a long time, often as long as 10 to 12 hours on a typical mainframe computer. Therefore, it's not practical to perform such a survey with any great frequency. As a result, survey information, as recorded in the SDB, tends to become somewhat out of date. For example, if a new module is added, deleted, or replaced (with a newer copy); or an entire product is added to (or deleted from) an existing library or new library, this will not be known to the Software Auditing System (SAS)

until the SURVEYOR is executed again, perhaps according to some fixed schedule, such as weekly or monthly, or simply when someone gets around to it.

The present invention, portions of which are illustrated in the annexed Figures 1 through 4, provides several techniques that represent improvements over the above process.

SUMMARY OF THE INVENTION

Accordingly, it is an object of the present invention to provide a software inventorying system which imposes a lower overhead on system resources.

It is another object of the present invention to provide an inventory system for software products which is substantially current at all times.

It is still a further object of the invention to provide a surveying system for software which reduces the overhead of monitoring software on a system.

The foregoing and other objects of the invention are realized by the present invention which provides a continuous software module surveyor which carries out the surveying of software modules on a continuous, rather than ad hoc, long periodic basis. To reduce software overhead and to increase the accuracy and reliability of the surveying/inventorying of the software, a continuous surveyor of the present invention contains a database of software modules on the computer system which is continuously updated by

detecting additions, deletions or modifications to software on an ongoing basis.

The surveying function of the present invention is carried out directly by surveying a library where the names of software modules are stored, or by monitoring the execution of modules and noting when a module is being executed which has not previously been in the system and/or by dynamic detection of software programs which carry out the tasks of installing, uninstalling, or modifying the locations where software modules are located.

Other features and advantages of the present invention will become apparent from the following description of the invention which refers to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a main flow chart of the surveying process of the present invention.

Figure 2 is a flow chart of surveying by monitoring the execution of software modules.

Figure 3 illustrates a search tree approach to finding module names in very large libraries/databases.

Figure 4 is modeled after Figure 3 and shows an aspect of a software module search tree.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

In general, the present invention is concerned with providing a continuously updated inventory (or list) of software modules comprising the software products in a computer in a substantially automated manner. The term computer, as used herein, denotes broadly, a single or a plurality of computers operating as a system or as a data center or the like. A computer can be a mainframe computer or it may be a network of PCs and it may use any of the available operating systems. Further, when the invention refers to the inventorying of software modules --sometimes using the term "surveying"-- it connotes and conveys the taking of the inventory of the software modules which means that the objective is to create a fairly accurate list of the software modules on the computer. However, it is virtually impossible to have an inventory so complete as to achieve anywhere near a 100% listing of all of the load modules on an active computer system. This is all the more so in very large mainframe computer centers where there are hundreds of software products comprising hundreds of thousands of load modules, sometimes approaching one million or more modules. Rather, taking an inventory means performing a substantial inventory of the software which would mean that at least 60 to 70 percent of the load modules have been inventoried. Moreover, when the invention refers to performing an

inventory in an automatic or substantially automatic manner, the intent again, is not to achieve 100% automation, which is virtually impossible. Rather, the achievement of the cataloging of on the order of about 60 to 70 or preferably 70-80% of the load modules constitutes carrying out the steps of the invention in an automatic or substantially automatic manner.

As described in this disclosure, the term "continuous" surveying needs to be distinguished from the term periodic surveying. In the prior art, the inventors herein are aware of scheduled surveying of load modules on large computer systems which typically are carried out at frequencies of typically a year or half a year or perhaps every three months or monthly. Given the great expense and computer overhead involved in taking a complete inventory of software modules on a large computer system, it is burdensome to perform such inventorying or surveying more frequently than every month or so. Therefore, the term continuous surveying as used herein means at least one or more of the following. It includes a system where the process of surveying is ongoing, meaning that as soon as a survey has been completed the next one is immediately initiated on an ongoing basis. The inventorying process is also continuous when it is performed periodically, the period being selected to be somewhat longer than the maximum time for conducting a survey.

For example, if a survey requires about two hours to complete, the continuous survey may be initiated every three hours. Furthermore, the definition of "continuous" surveying includes surveys that are carried out at least twice as frequently as any prior art scheduled surveys. Another example of "continuous" surveying is inventorying of software that is initiated or triggered automatically in response to the detection of the execution of software modules not listed in an existing module inventory.

Described below are several techniques for maintaining updated inventory lists, including the ones referred to below as continuous surveying, surveying during monitoring, and dynamic detection of module installation or change.

Continuous SURVEYOR (CS): Rather than perform the Survey function periodically, the CS is a continuous activity that creates and maintains the SDB (Figure 1), ensuring that the SDB is kept current. Moreover, the CS minimizes or eliminates unnecessary module analysis activity by attempting to only survey those modules that are new or have changed since the last time they were subject to module analysis.

The CS creates and uses Module Indicators (MI) and Library Indicators (LI):

- An MI is associated with a particular module in a particular library of the computer system being surveyed, and incorporates, or

is derived from, certain information about the module, all of which is available in the library's directory, though the precise nature and extent of this information will differ from operating system to operating system. (The "directory" of a library is an index or card-catalog to the modules contained in the library. The directory can be efficiently read, in its entirety, to gain overview information about the entire contents of the library.) The characteristics which (subject to their availability in the particular operating system) the CS incorporates into each MI would typically include the module size, the date of creation, and the address or location within the containing library where the module is located. Note that one or more of these characteristics will typically change if the module is replaced with a different version by the same name. For example, if a new version of the software product to which the module belongs were to be installed on the same library, the creation date of the new module would certainly differ, the size of the new module would very likely have changed, as would its storage location. The MI is created by

treating the module directory data in any of the following ways: as a single numerical value; mathematically transformed into a numerical value such as a computed checksum, hash total, CRC total or the application of other such mathematical calculations wherein the probability of detecting any change of one or more module characteristics is at least 90%; or arranged as a pre-determined sequence of bytes. The essential characteristic of the MI is that it is highly probabilistic that it will have a different value if one or more of the constituent module characteristics changes in value.

- An LI is derived from the set of MIs pertaining to all the modules residing on the library. In one embodiment, the LI consists of a hash total of all the MIs, consisting of an arithmetic sum of all the MIs, treating each as a numeric value. In another embodiment, an LI is derived from the set of MIs as a calculation such as a computed checksum, hash total, CRC total or the application of other such mathematical calculations wherein the probability of detecting any change of one or more MIs is at least 90%. The essential characteristic

of the LI is that it is highly probabilistic that it will have a different value if one or more of the constituent MIs changes in value.

LIs and MIs are retained in a database, and are associated with the respective libraries and modules in the SDB to which they pertain. Historical data regarding the LIs and MIs consisting of at least one prior value is also retained either in the database or as a separate file or database representing a prior inventory. The preferred technique for accomplishing this is to store the LI and MI data directly in the SDB, as part of the record associated with each library or module.

CS processing, as shown in Fig. 1, is as follows:

1. Point to the library on the system. (If the CS is presently pointing to the last library on the system, start over by pointing to the first library, step 12.) Examine the library's directory and determine a current-MI for each module found in the library's directory. From this set of current-MIs, determine a current-LI for the library.

2. Compare the current-LI to the LI in the SDB that corresponds to the current library, derived at the time a module-analysis of one or more modules on this library was last performed. If the current-LI is identical to the LI in the SDB, then no modules on the library have been added, deleted or modified since

they were last surveyed, in which case, processing continues at step 6. (Steps 14, 16).

3. As shown in steps 18-38, compare in turn, each current-MI for each module on the current library with the prior version of the MI for that module as stored in the SDB. If no prior version of the MI exists (indicating that the module has been added since the last survey), or if the two MIs differ (indicating that the module has been replaced or updated), perform a module-analysis of the module and store the module-analysis information in the SDB together with the module's current-MI and the date of the survey activity.

4. If, after all modules on the current library have been processed, a prior MI exists in the SDB for a module that was not found in the current library, mark the module and the MI in the SDB as "deleted", along with the date of the survey.

5. If, after all modules on the current library have been processed, any modules have been subjected to module-analysis or have been found to have been deleted, record the current-LI (optionally with the date of the survey) in the SDB.

6. Optionally, in order to spread out the overhead of the surveying activity, defer proceeding for a brief interval. For example, this interval might be calibrated to complete an entire cycle of the libraries in some specified approximate time, such as

an hour.

7. Continue at step 1.

Note that when the CS is initiated for the first time, the SDB is empty, and therefore every module on every library on the system will be subject to module analysis, as in the batch SURVEYOR. Thereafter, however, the need to perform module analysis on new or changed modules will be relatively infrequent, while still ensuring that the surveyed information is always up-to-date.

Surveying during Monitoring: As an independent notion from the CS above, and as shown in Fig. 2, steps 40-48, the MONITOR process is enhanced to recognize when a "new" module is being executed, that is, a module recently installed on the system, and therefore not currently recorded in the SDB, and perform the required module-analysis of that module immediately. Optionally, module-analysis is performed on every module on the same library as the "new" module, on the theory that new modules are associated with the installation of new software, which likely contains other modules as well. The process is as follows:

1. Each time the MONITOR detects the execution of a module, the MONITOR accesses the current SDB to determine if a module with the same name as the executed module is recorded, and on the same library as the executed module. If so, the module was

previously surveyed, and MONITOR processing continues as normal. If the executed module is not found in the SDB, processing continues with step 2.

2. The MONITOR initiates an immediate module-analysis of the module. The information about the module gathered by the module-analysis process is either a) used to update the SDB directly, or b) included, as a distinct record-type, in the execution and usage data that the MONITOR emits. This latter approach would be preferred if the SDB to which the MONITOR has access is a read-only version of the SDB, or if it is an in-storage representation on the SDB, and therefore can't be updated directly.

As an alternative to the above process, the MONITOR can initiate or notify another process (e.g., the CS or SURVEYOR) to perform the module-analysis of the executed module or of the entire library.

As a variant of the above process, when the MONITOR detects a newly-installed module it optionally surveys the entire library containing that module, on the theory that other related modules may have been installed as well.

As another variant, the MONITOR creates an MI of the executing module. If the MI does not match the MI for that module on file in the SDB, then the module has been updated, replaced, or otherwise changed from the one previously surveyed. If so, the process proceeds as in step 2 above.

As yet another variant, the MONITOR creates an MI of the executing module. If the MI matches the MI for a module on file in the SDB having the same name but located on a different library, then the module is most likely a copy of the one detected by the CS. If so, the process proceeds to survey the module and its library as in step 2 above.

As another variant, the SDB, which would typically exist as a database on external disk storage, is represented as a memory resident table, which will improve the MONITOR's speed and efficiency by eliminating the need to perform time consuming I/O when looking up each module that's executed.

As there can be hundreds of thousands of modules, the use of a memory resident table can present some challenges especially since many modules have similar names (such as IEFGTR13, IEFGTR14, IEFGTRMS). An efficient means of implementing this table is to use a form of binary search tree, and a ternary search tree, in particular. Ternary search trees combine the time efficiency of digital search trees with the space efficiency of binary search trees.

There are numerous ways in which module names can be organized in such a tree structure. In one implementation, a list of library names is constructed with each element in the list referencing a ternary tree of module names, each of which is contained within that library (Figure 3). A primary advantage of

this organization is that the MONITOR can reduce memory requirements by selectively loading those portions of the SDB pertaining to libraries which have been checked for names of executing modules. For example, little used or archival libraries can be excluded from consideration by the MONITOR.

In another implementation, the SDB is organized in memory as a single ternary tree of all known module names on the system. The last element in each tree node (at which point the module name is complete) contains (or references) a list of all libraries in which this module (and copies of this module) is located (Figure 4). In the example shown, a module name ending in "13", perhaps IEFGTR13, is only found in LIBC, however, the module name ending in "14" (e.g., IEFGTR14) is known to be in two libraries: LIBA and SYSLIB1.

In another implementation, the SDB is organized in memory as a table indexed by the hash code of the module names found on the system. A primary advantage of this method is that number of operations required to locate a record in memory is both constant and fast.

Dynamic Detection of Module Installation or Change: As an independent notion from the techniques described above, a Dynamic Detection process directly detects the act of adding, deleting, copying or modifying one or more modules on the system, or the

act of creating a new library, or moving, copying or renaming an existing library, and triggers the surveying process immediately, and only for the added, deleted, copied or changed modules or libraries. The Dynamic Detector (DD) is a continuously running process (which could optionally be combined with another such process, such as the CS or MONITOR) that establishes certain intercepts or hooks into certain operating system functions so that when any modules are added, deleted or modified within the system, in any of the several ways in which this could occur, the DD receives control, and is able to perform analysis and record information. For example, this may be accomplished in OS/390 by intercepting the BLDL function, which is invoked when a module is written into a load library. When a module is subsequently added, deleted or modified, the DD receives control and either performs a module-analysis itself, or invokes another process, such as the Surveyor, to do so, or signals a continuous process, such as the CS, to do so.

The benefits of employing any of these methods are numerous:

The entire computer system can be surveyed without imposing any undo overhead;

Continuous surveying ensures that data will be kept up-to-date;

Monitoring of executing modules can determine

modules not previously encountered by surveying;

Monitoring of executing modules can detect modules that have changed from that encountered in the last survey;

Monitoring of executing modules can provide an indicator that other modules in the same library are likely to have changed;

Monitoring of executing modules provides a convenient mechanism to signal that a survey should be conducted.

Monitoring of program usage can be carried out more efficiently in that the inventory of load modules is more up to date and accurate. In the prior art, monitoring of usage was typically done over a period of from six weeks to three months during which time the inventory of the software may have changed considerably, creating various ambiguities and difficulties. This is avoided by the present invention.

By carrying out inventorying/surveying of software modules virtually immediately after they are installed, it is easier to identify the software products to which they are associated for at least two main reasons. The first is that there is still knowledge of the personnel who installed them so that they can immediately also indicate to the surveying program the software product name, resolving ambiguities quickly. Secondly, when a new software is

installed, the name of the software product is typically also loaded in a special library and that library can be consulted to immediately determine the corresponding software names which are associated with specific load module names.

Although the present invention has been described in relation to particular embodiments thereof, many other variations and modifications and other uses will become apparent to those skilled in the art. It is preferred, therefore, that the present invention be limited not by the specific disclosure herein, but only by the appended claims.

WHAT IS CLAIMED IS:

1. Apparatus for inventorying software modules related to software products stored on a computer, the apparatus comprising:

a database storing a module name inventory representing an inventory of software modules stored on the computer; and

a first software program that continuously inventories the computer to maintain the module name inventory in the database current.

2. The apparatus of claim 1, in which the first software program inventories the modules by monitoring required changes to the database relative to a prior inventory, the changes comprising required additions or deletions of module names and/or in which the first software program takes inventory of the software modules by monitoring execution of load modules and/or in which the inventorying of software modules is carried out by monitoring the operation of software steps within the computer which handle the addition or deletion or modification of the contents or locations of software modules and/or by dynamic detection of module installation or change and each such detection triggers a surveying process for the software module.

3. The apparatus of claim 2, in which the

monitoring of the changes is carried out relative to module libraries in which the module names are stored and/or at a frequency of at least once daily.

4. The apparatus of claim 1, including a second software program which converts the list of inventoried software modules to a list of corresponding software products stored on the computer.

5. The apparatus of claim 2, in which the computer is a mainframe computer or a network of PCs.

6. The apparatus of claim 2, in which the first software program creates and uses module indicators and library indicators, and the module indicators and library indicators are used to create and maintain the inventory of software modules.

7. The apparatus of claim 6, in which the library indicators comprise a mathematical calculation performed relative to the module indicators and/or in which the library indicators and the module indicators are stored directly in the database.

8. The apparatus of claim 2, in which the inventorying of software modules is carried out substantially automatically.

9. The apparatus of claim 2, in which the first software program includes execution monitoring software which detects a change to the inventory by monitoring execution of modules and which triggers a survey of a module library in which a change has been detected.

10. The apparatus of claim 9, in which the execution monitoring software locates module data in the database by use of a hash table and/or in which the execution monitoring software locates module data in the database by reference to a search tree.

11. The apparatus of claim 10, in which the search tree is binary or ternary.

12. The apparatus of claim 9, in which modules are compared to one another through a comparison of their contents.

13. The apparatus of claim 2, in which the database includes a field which comprises a fingerprint of a corresponding module.

14. The apparatus of claim 13, in which the fingerprint includes one or more of: the size of the module, the name and size of one or more constituent set or parts of the module, the dates that one or more

constituent parts was compiled, the date that the constituent parts were link edited or bound together to create the module, and textual data contained within the module.

15. The apparatus of claim 2, in which the dynamic detection is performed by including intercepts or hooks into functions of the operating system which deal with the addition, deletion or modification of modules.

16. A method for inventorying software modules related to software products stored on a computer, the method comprising the steps of:

storing a module name inventory representing an inventory of software modules stored on the computer in a database; and

continuously inventorying the computer to maintain the module name inventory in the database current.

17. The method of claim 16, including inventorying the modules by monitoring required changes to the database relative to a prior inventory, the changes comprising required additions or deletions of module names and/or inventorying the module names at a frequency of at least once daily and/or inventorying by monitoring execution of load modules

and/or inventorying the software modules by monitoring the operation of software steps within the computer which software steps handle the addition or deletion or modification of the contents or locations of software modules and/or inventorying the software modules by dynamic detection of module installation or change and employing the detection of module installation or change to trigger a surveying of the software modules.

18. The method of claim 17, including detecting the changes relative to module libraries in which the module names are stored.

19. The method of claim 16, including converting a list of inventoried software modules to a list of corresponding software products stored on the computer.

20. The method of claim 17, including creating and using module indicators and library indicators, the module indicators and library indicators being used to create and maintain the inventory of software modules.

21. The method of claim 20, in which the library indicators comprise a mathematical calculation performed relative to the module indicators.

22. The method of claim 16, including inventorying the software modules substantially automatically.

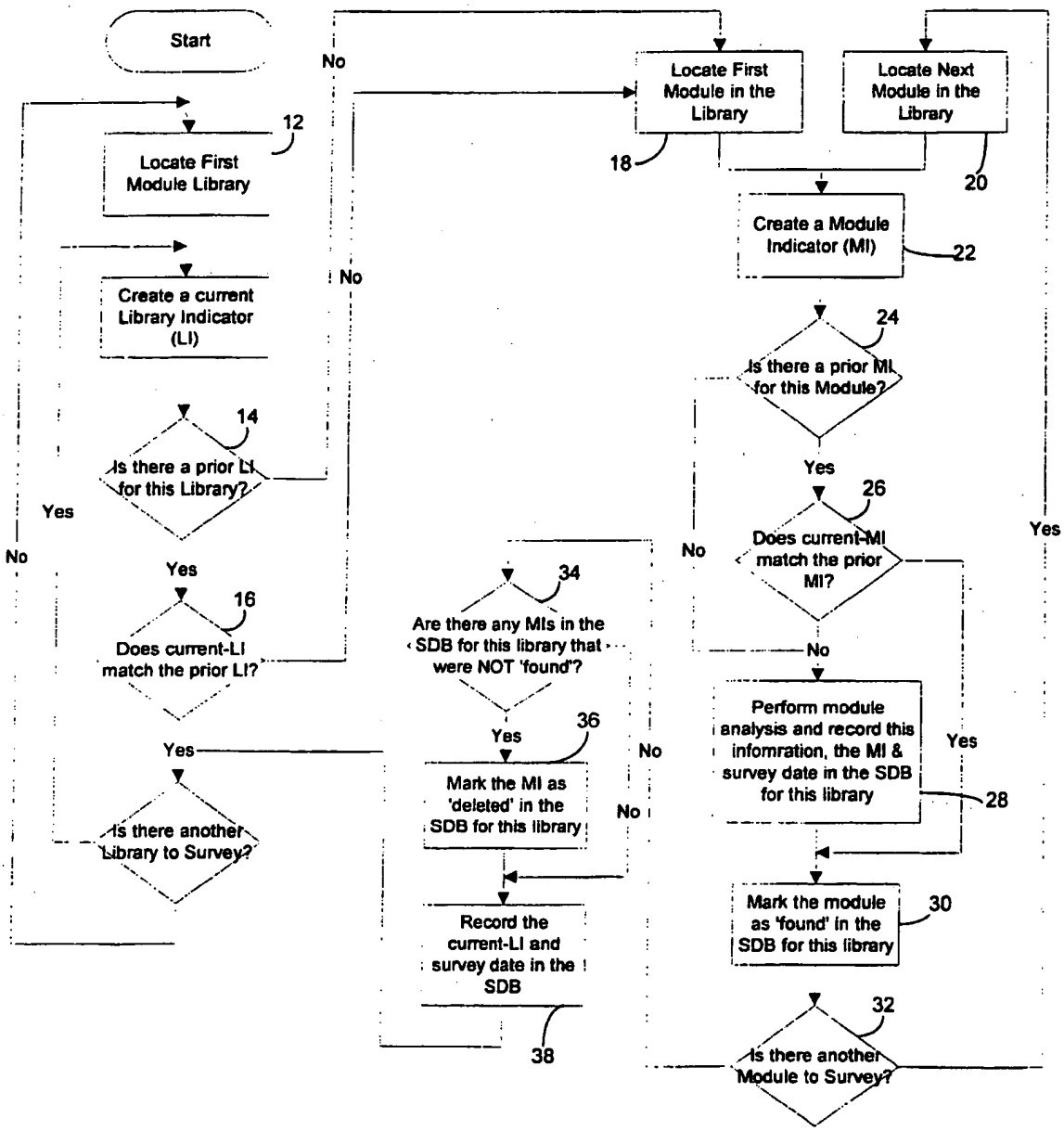
23. The method of claim 17, including detecting changes to the inventory by monitoring execution of modules and triggering a survey of a module library in which a change has been detected.

24. The method of claim 23, including locating module names by reference to a search tree.

25. The method of claim 23, including comparing modules to one another to obtain a comparison of their contents.

26. The method of claim 17, including storing in the database a field which comprises a fingerprint of corresponding modules.

27. The method of claim 17, including performing the dynamic detection by including intercepts or hooks into functions of the operating system which are associated with the addition, deletion or modification of modules.


Start
Locate First Module Library
12
Create a current Library Indicator (LI)
Is there a prior LI for this Library?
14
Does current-LI match the prior LI?
16
Is there another Library to Survey?
Yes
No
Locate First Module in the Library
18
Locate Next Module in the Library
20
Create a Module Indicator (MI)
22
Is there a prior MI for this Module?
24
Does current-MI match the prior MI?
26
Perform module analysis and record this infomration, the MI & survey date in the SDB for this library
28
Mark the module as 'found' in the SDB for this library
30
Is there another Module to Survey?
32
Are there any MIs in the SDB for this library that were NOT 'found'?
34
Mark the MI as 'deleted' in the SDB for this library
36
Record the current-LI and survey date in the SDB
38

Figur 1

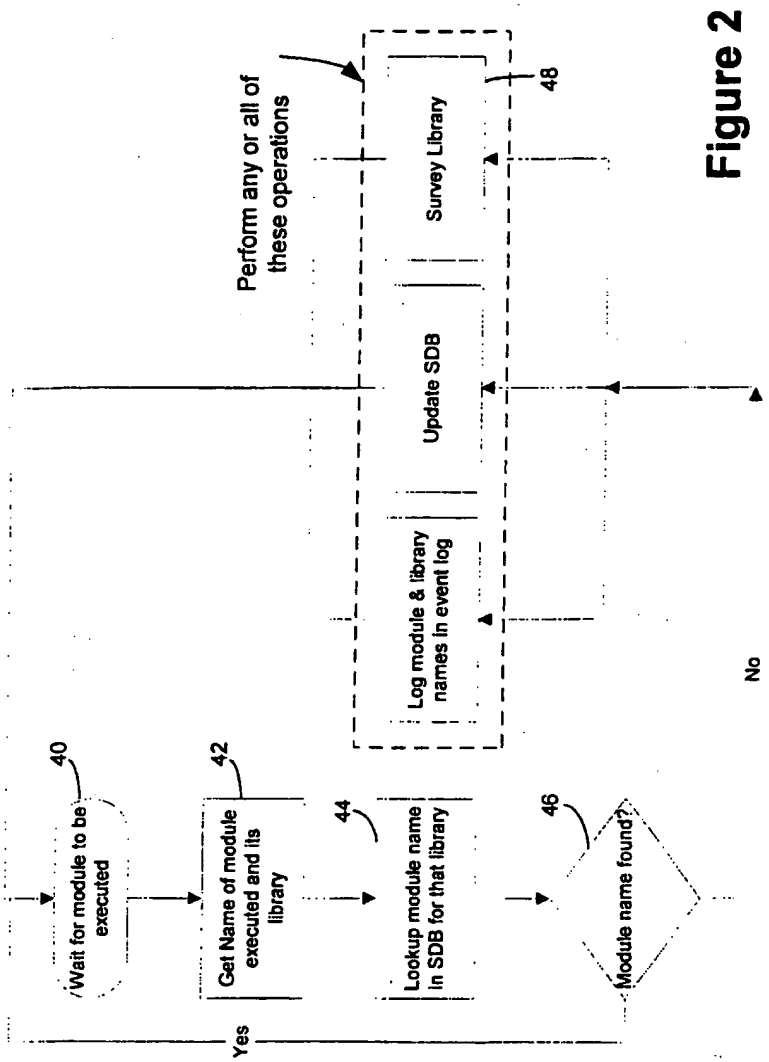

Perform any or all of these operations
Survey Library
48
Update SDB
Log module & library names in event log
40
Wait for module to be executed
42
Get Name of module executed and its library
44
Lookup module name in SDB for that library
46
Module name found?
Yes
No

Figure 2

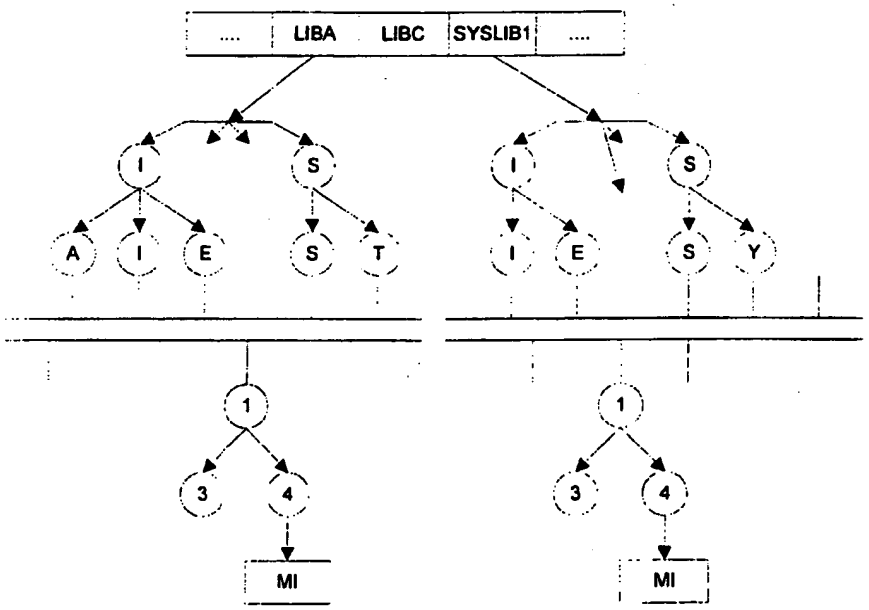


Figure 3

Figure 4